// HDFSReader.java

import java.io.BufferedOutputStream;

import java.io.FileOutputStream;

import java.io.InputStream;

import java.io.OutputStream;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IOUtils;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class HDFSReader extends Configured implements Tool {

public static final String FS\_PARAM\_NAME = "fs.defaultFS";

public int run(String[] args) throws Exception {

if (args.length < 2) {

System.err.println("HdfsReader [hdfs input path] [local output path]");

return 1;

}

Path inputPath = new Path(args[0]);

String localOutputPath = args[1];

Configuration conf = getConf();

System.out.println("configured filesystem = " + conf.get(FS\_PARAM\_NAME));

FileSystem fs = FileSystem.get(conf);

InputStream is = fs.open(inputPath);

OutputStream os = new BufferedOutputStream(new FileOutputStream(localOutputPath));

IOUtils.copyBytes(is, os, conf);

return 0;

}

public static void main(String[] args) throws Exception {

int returnCode = ToolRunner.run(new HDFSReader(), args);

System.exit(returnCode);

}

}

// HDFSwriter.java

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.util.Tool;

import java.io.BufferedInputStream;

import java.io.FileInputStream;

import java.io.InputStream;

import java.io.OutputStream;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IOUtils;

import org.apache.hadoop.util.ToolRunner;

public class HDFSwriter extends Configured implements Tool {

public static final String FS\_PARAM\_NAME = "fs.defaultFS";

public int run(String[] args) throws Exception {

if (args.length < 2) {

System.err.println("HdfsWriter [local input path] [hdfs output path]");

return 1;

}

String localInputPath = args[0];

Path outputPath = new Path(args[1]);

Configuration conf = getConf();

System.out.println("configured filesystem = " + conf.get(FS\_PARAM\_NAME));

FileSystem fs = FileSystem.get(conf);

if (fs.exists(outputPath)) {

System.err.println("output path exists");

return 1;

}

OutputStream os = fs.create(outputPath);

InputStream is = new BufferedInputStream(new FileInputStream(localInputPath));

IOUtils.copyBytes(is, os, conf);

return 0;

}

public static void main(String[] args) throws Exception {

int returnCode = ToolRunner.run(new HDFSwriter(), args);

System.exit(returnCode);

}

}

// WCMapper.java

package wordcount;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter rep) throws IOException

{

String line = value.toString();

for (String word : line.split(" "))

{

if (word.length() > 0)

{

output.collect(new Text(word), new IntWritable(1));

}

}

}

}

// WCReducer.java

package wordcount;

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterator<IntWritable> value, OutputCollector<Text, IntWritable> output, Reporter rep) throws IOException

{

int count = 0;

while (value.hasNext())

{

IntWritable i = value.next();

count += i.get();

}

output.collect(key, new IntWritable(count));

}

}

// WCDriver.java

package wordcount;

import java.io.IOException;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class WCDriver extends Configured implements Tool {

public int run(String args[]) throws IOException

{

if (args.length < 2)

{

System.out.println("Please give valid inputs");

return -1;

}

JobConf conf = new JobConf(WCDriver.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

conf.setMapperClass(WCMapper.class);

conf.setReducerClass(WCReducer.class);

conf.setMapOutputKeyClass(Text.class);

conf.setMapOutputValueClass(IntWritable.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);

return 0;

}

public static void main(String args[]) throws Exception

{

int exitCode = ToolRunner.run(new WCDriver(), args);

System.out.println(exitCode);

}

}

// WeatherMapper.java

package weather;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WeatherMapper extends Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text hotOrCold = new Text();

@Override

protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] tokens = value.toString().split("\\s+");

if (tokens.length >= 2) {

float temperature = Float.parseFloat(tokens[1]);

if (temperature > 30.0) {

hotOrCold.set("Hot");

} else if (temperature < 10.0) {

hotOrCold.set("Cold");

} else {

return;

}

context.write(hotOrCold, one);

}

}

}

// WeatherReducer.java

package weather;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class WeatherReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

context.write(key, new IntWritable(sum));

}

}

// WeatherDriver.java

package weather;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WeatherDriver {

public static void main(String[] args) throws Exception {

if (args.length != 2) {

System.err.println("Usage: WeatherDriver <input path> <output path>");

System.exit(-1);

}

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Weather Data Analysis");

job.setJarByClass(WeatherDriver.class);

job.setMapperClass(WeatherMapper.class);

job.setReducerClass(WeatherReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

// MatrixMultiplicationMapper.java

package matrix;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MatrixMultiplicationMapper extends Mapper<Object, Text, Text, Text> {

@Override

protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] parts = value.toString().split(",");

String matrixName = parts[0];

int row = Integer.parseInt(parts[1]);

int col = Integer.parseInt(parts[2]);

double matrixValue = Double.parseDouble(parts[3]);

if (matrixName.equals("A")) {

for (int k = 0; k < MatrixMultiplicationDriver.NUM\_COLUMNS\_B; k++) {

context.write(new Text(row + "," + k), new Text("A," + col + "," + matrixValue));

}

} else if (matrixName.equals("B")) {

for (int i = 0; i < MatrixMultiplicationDriver.NUM\_ROWS\_A; i++) {

context.write(new Text(i + "," + col), new Text("B," + row + "," + matrixValue));

}

}

}

}

// MatrixMultiplicationReducer.java

package matrix;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

public class MatrixMultiplicationReducer extends Reducer<Text, Text, Text, Text> {

@Override

protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

Map<Integer, Double> A\_elements = new HashMap<>();

Map<Integer, Double> B\_elements = new HashMap<>();

for (Text value : values) {

String[] parts = value.toString().split(",");

String matrixName = parts[0];

int index = Integer.parseInt(parts[1]);

double matrixValue = Double.parseDouble(parts[2]);

if (matrixName.equals("A")) {

A\_elements.put(index, matrixValue);

} else if (matrixName.equals("B")) {

B\_elements.put(index, matrixValue);

}

}

double sum = 0.0;

for (int j : A\_elements.keySet()) {

if (B\_elements.containsKey(j)) {

sum += A\_elements.get(j) \* B\_elements.get(j);

}

}

context.write(key, new Text(Double.toString(sum)));

}

}

// MatrixMultiplicationDriver.java

package matrix;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MatrixMultiplicationDriver {

public static final int NUM\_ROWS\_A = 2;

public static final int NUM\_COLUMNS\_B = 2;

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Matrix Multiplication");

job.setJarByClass(MatrixMultiplicationDriver.class);

job.setMapperClass(MatrixMultiplicationMapper.class);

job.setReducerClass(MatrixMultiplicationReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}